UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

INGENIERIA CIVIL INFORMATICA

MIGRACIÓN DE ARQUITECTURA MONOLÍTICA, A UNA DISTRIBUIDA A TRAVÉS DE DDD Y EDA PARA MEJORAR PERFORMANCE Y ESCALABILIDAD

Memoria de Título presentada a la Facultad de ingeniería de la Universidad de Concepción para optar al grado de ingeniero civil informático

POR: Vicente Nicolás Herrera Jiménez

Profesor Guía: Geoffrey Hecht

Ingeniero Supervisor: Rodrigo Cortés

Concepción, Chile 2024

Este es el inicio del final para esta etapa universitaria y no cabe duda de que debo reconocer y agradecer el apoyo incondicional de mis padres durante este proceso. A mis amigos de toda la vida quienes me vieron y me aconsejaron en mis momentos de flaqueza. Y, por último, pero no menos importante al tremendo grupo humano al cual también puedo llamar amigos y compañeros de estudio con los cuales he coincidido a lo largo de mi estancia en esta universidad. Quienes a pesar de las diferentes e inesperadas circunstancias en las cuales se desarrolló nuestra etapa universitaria, se mantuvieron como un importante pilar de apoyo y motivación. Son el vivo ejemplo de una camaradería para muchos envidiable y una calidad humana excepcional.

Vicente.

Resumen

Blue Express es la empresa líder de logística y distribución en Chile perteneciente al grupo COPEC la cual tiene como misión ser el impulsor del crecimiento en los negocios de las personas para llevarlas al siguiente nivel a través de la logística e-commerce.

Dentro de la compañía se está viviendo un proceso de transformación tecnológica la cual requiere cumplir con los requerimientos presentes y futuros de demanda comercial y tecnológica que requiera la operación de la compañía.

En esta memoria se analizarán diferentes aspectos del proceso de diseño, implementación y puesta en producción de diferentes componentes de software.

Estos son 3 APIs y un consumidor los cuales participarán de manera integral en el funcionamiento operativo de la compañía, se evaluarán, se probarán y serán colocados en producción. Todo esto bajo un marco de Domain Driven Design, las 3 APIs están en el dominio de órdenes y el consumidor dentro de tracking operacional mostrando a la vez él uso de una arquitectura basada en eventos (EDA).

Finalmente se vio la implementación de estos artefactos y sus resultados de desempeño en producción siendo estos últimos satisfactorios.

Tabla de Contenidos

1. Introducción

Blue Express presenta el inicio de su historia en el año 1996, como una filial de Lan Airlines, llamada Lan Courier, donde posteriormente el año 2001 comienzan con las operaciones de almacenaje en su nuevo centro de distribución.

En el año 2008 se establece el cambio de marca a Blue Express.

Blue Express tiene como propósito impulsar la logística E-commerce para que cada día sea más accesible para todos [1]. Blue cuenta con diferentes servicios para sus clientes tales como: Fulfillment, PYME, Empresa, Flex e integración E-commerce.

En la actualidad Blue Express cuenta con la red de envíos más amplia a nivel nacional y la mayor cantidad de paquetes procesados anualmente en el país.

1.1. Antecedentes Generales del Problema

Hoy en día en Blue Express cuenta con sistemas que funcionan en el ecosistema de una gran base de datos de Oracle que data de tiempos antiguos de la compañía, en esta base de datos se manejan diferentes procesos de negocio de variados temas y áreas tales como, facturación, seguimiento, documentación, diferentes servicios e incluso la impresión de etiquetas.

Durante mucho tiempo la arquitectura sólo dependía de este monolito alojado en la BBDD de Oracle, la cual cumplía con los requerimientos del negocio de la compañía, las capacidades del motor de la base eran suficientes para las demandas que tenía Blue Express.

Sin embargo, los desarrollos de funcionalidades sobre este motor estaban externalizados con distintas empresas de consultoría, lo cual a largo plazo también afectó a la mantenibilidad de la base de código ya que cada consultora creaba una funcionalidad (con su propio estilo) y si pasaba a romper otra de otro equipo era difícil saberlo.

Posteriormente con la explosión del E-commerce [2] y con la aún más rápida adopción de este durante el tiempo de la pandemia del Covid-19, se detectaron y analizaron varias falencias y denominados cuellos de botella que se generaban en la operación de la compañía a raíz de este sistema de información antes descrito los cuales implicaban problemas de bloqueos por el orden de las operaciones o demoras en el procesamiento de solicitudes.

Es importante recordar que hasta ese momento la empresa siempre había contado con un enfoque mucho más centrado en lo operacional que en lo tecnológico, entonces la inversión en tecnología y en desarrollos se encontraba en un segundo plano frente a lo operacional.

Estos cuellos de botella consisten en varias problemáticas que se evidenciaron con el pasar del tiempo, ejemplo de esto son inestabilidades en el servicio, una baja de rendimiento en operaciones de mediano y gran tamaño, downtime generalizado con afectación a todo el proceso, algunos usos de la base de datos deben restringirse por bloquear las tablas core o

incluso las soluciones temporales fueron insuficientes tales como el procesamiento por batches.

Esto se debe a que todo lo que tiene que ver con procesos de escritura de información, va a la misma base de datos, este es un problema inherente del monolito y a medida que escala el tamaño de la compañía, la arquitectura queda más y más atrás debido a su problema con la escalabilidad.

Con todo lo antes expuesto, es que durante los últimos años se estaba evaluando la reformulación del departamento de tecnologías de la información de la compañía, con tal de tomar mayor propiedad sobre los desarrollos hechos por y para la empresa, además de poder imponer diferentes aristas de estandarización y buenas prácticas en el desarrollo de tecnologías.

Esto llevó al diseño de una migración progresiva del monolito, por lo tanto este seguirá siendo utilizado durante el proceso de modernización, se aislarán funcionalidades y procesos de negocio poco a poco, creando información nueva en el modelo propuesto y replicando esa información al sistema de legado, hasta que ya se pueda dejar de depender de este último.

1.2. Antecedentes Generales de la Solución

Antes de ver la solución propuesta es importante comprender ciertos conceptos relevantes y lo que ellos contienen e implican para el desarrollo de esta migración.

Para comenzar se explicará lo que es el diseño guiado por el dominio (DDD por sus siglas en inglés).

El DDD es una metodología de desarrollo de software que se enfoca en acercar el negocio al código a través del uso de un lenguaje descriptivo para clases, métodos y variables, este lenguaje es acordado entre los equipos de desarrollo, los de negocio y en algunos casos los usuarios, generándose lo que se denomina “Lenguaje Ubicuo”. Esto último es especialmente importante en el caso de la compañía ya que esta no es una compañía que esté internalizada en los procesos de desarrollo, entonces esto facilita una mejor comunicación para la migración y para el funcionamiento a posterior.

Adicionalmente al momento de programar, a los repositorios se les definió una estructura en base al DDD, esto es una manera de organizar las dependencias dentro del repositorio mismo, por consiguiente, dentro de la capa de Domain(Dominio) solamente sabe de sí misma, la capa de Application(Aplicación) sabe sobre sí misma y sobre la de Dominio, esto se repite hacia las capas exteriores.

Entonces por ejemplo en la capa de dominio encontraremos los archivos que en NestJS se denominan servicios, en la capa de aplicación los controladores y en la capa de infraestructura las conexiones a las bases de datos.

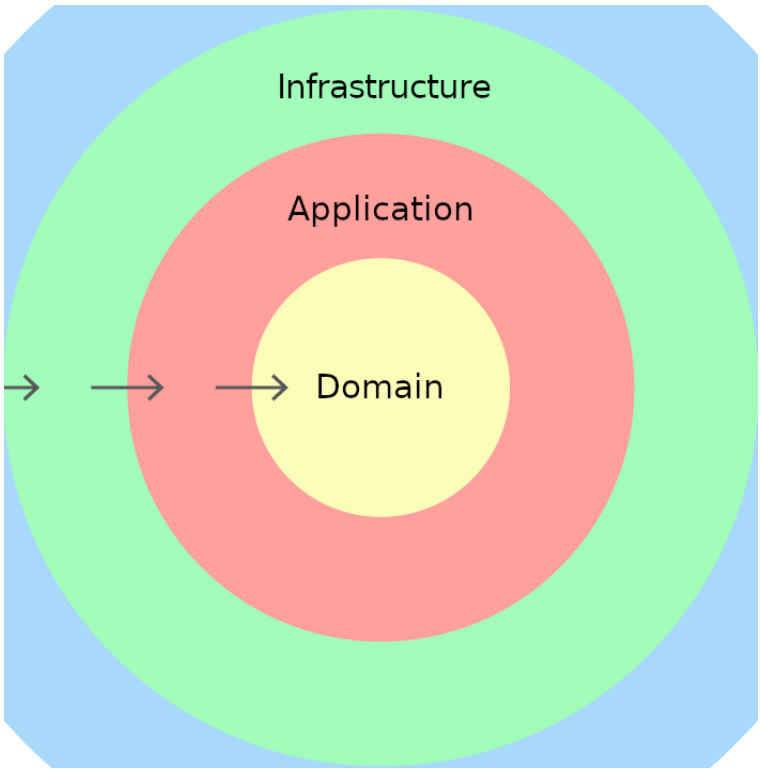


Figura 12: Diagrama de la división de capas en base a DDD.

Esto también se cumple para el consumidor que se desarrolla, incluso si es en otro lenguaje ya que la filosofía del DDD es agnóstica a ellos y permite mantener una estructura uniforme del código para las diferentes tecnologías y artefactos logrando de esta manera que sea más fácil crear nuevos repositorios y adaptarse a ellos para los desarrolladores que posean menos contexto.

Y hablando de consumidores es importante desarrollar el concepto de Arquitectura Basada en Eventos (EDA por sus siglas en inglés). Tal como dice el nombre, esta arquitectura es un paradigma de programación, el cual se centra en los **eventos** y para definir estos, se puede resumir en que son cambios significativos de estado. Esto se puede traducir a los movimientos de carga en el caso de la operación de Blue, a modo de ejemplo, un paquete cuando ingresa al sistema desencadena un evento, cuando este sale a reparto es otro nuevo evento y cuando ya es recibido por el destinatario es otro evento.

Los eventos necesitan tener un recorrido a nivel de sistema y ahí es donde entra EDA. Si bien se tiene una parte centrada en la orientación a servicios, esto no es excluyente con EDA, mas bien es complementario. Entonces en lo que a eventos se refiere, se define un flujo con diferentes partes, ella siendo por ejemplo:

- Productores de eventos
- Canal de eventos
- Consumidores de eventos

Un Servicio puede ser un productor de eventos, como puede ser el caso de un servicio que genera eventos sobre donde se encuentra el paquete a transportar.

El canal a utilizar en el caso de Blue, son los servicios de SNS [X] (Simple Notification Service) y SQS (Simple Queue Service) de AWS (Amazon Web Services) en donde encontramos Temas y Colas respectivamente.

Y un Consumidor es un artefacto el cual recibe los mensajes generados y en base a ellos se dispara una nueva acción.

1.3. Solución Propuesta y Alcances

Desde el departamento de arquitectura se ha propuesto pasar a un sistema distribuido con múltiples servicios y bases de datos de diferente índole, la división de responsabilidades se ha dado según el "dominio" que se le ha asignado a cada célula de desarrollo.

Para abordar el desarrollo de esta nueva arquitectura basada en eventos se ha definido que el monolito será estrangulado de manera progresiva. Para lograr estos se han definido diferentes hitos los cuales se van a cumplir con el pasar de los meses y de los desarrollos satisfactorios que generen los equipos.

Como esta tarea tiene dimensiones gigantescas por el tamaño de la compañía y la complejidad que presenta el proceso obviamente no se puede hacer todo el trabajo con solo una persona, así que en este trabajo se abordará la creación de diferentes aparatos de software que suplen necesidades específicas del área del ciclo de vida de la orden.

Para esto se ejecutara la creación de 3 servicios (APIs) y un consumidor de mensajes, estos se encontraran ceñidos a los requerimientos de calidad y estructuras que solicita el departamento de arquitectura de la empresa.

Se presentarán diferentes tipos de documentación, entre ellos diagramas, especificaciones según estándares de la industria y pruebas automatizadas de tipo unitarias.

1.4. Objetivo General

El objetivo general por cumplir es la de replicar o crear funcionalidades presentes en el monolito tales como la consulta y creación de ordenes de servicio además del seguimiento de estas, pero llevándolas al nuevo modelo, además de buscar una mejora de performance y escalabilidad.

1.3.1 Objetivos Específicos

a) Estudio sobre el funcionamiento de la compañía y análisis del proceso a mejorar
b) Diseño de las funcionalidades del aparato de software a desarrollar.
c) Creación de documentación acorde al servicio a desarrollar y compatible con el resto del sistema.
d) Desarrollo de los componentes previamente documentados.
e) Pruebas de rendimiento y funcionalidad del software desarrollado.

1.5. Metodología de Trabajo

Tal como lo pide el estándar de la compañía, las metodologías a usar durante el desarrollo son principalmente SCRUM [3] y el uso de un tablero Kanban [4] en Jira, en el caso específico del equipo de desarrollo los sprints tienen una duración de dos semanas donde se abordan diferentes historias de usuario las cuales pueden venir desde la persona que ejerce el rol de Product Owner o también de quien es Líder Técnico, ya que pueden incluir temas de mantención o performance.

1.6. Estructura del informe

En las próximas secciones se describirán diferentes etapas del proceso de desarrollo de algunos de los artefactos que son requeridos para avanzar en la migración hacia una arquitectura basada en eventos. A continuación, se detalla el contenido de cada una de ellas.

Sección 2: Análisis: En esta sección se revisan los requerimientos a cumplir del artefacto de software, en contexto sobre el sistema en donde se inserta y cómo se relaciona con el mismo. Además, en caso de ser necesario se agregan o detallan tareas para el desarrollo.

Sección 3: Diseño y Documentación: Esta sección describe las decisiones de diseño de la implementación y se genera la documentación necesaria según las directrices del departamento de arquitectura para proceder a la etapa de Desarrollo.

Sección 4: Desarrollo: En esta sección se mostrará más a detalle el cómo se avanza y se completa el desarrollo de los artefactos.

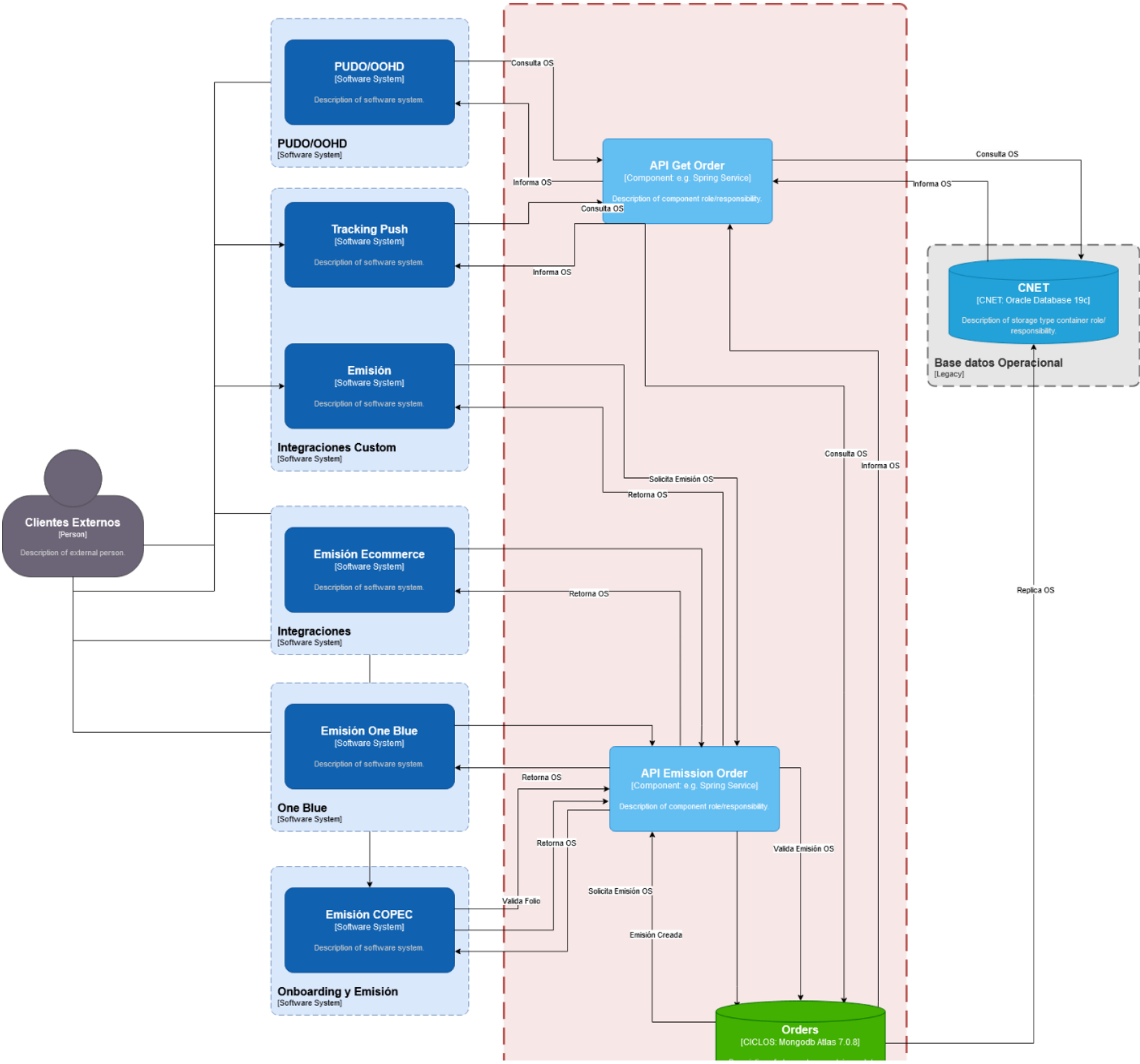
Sección 5: Evaluación: En esta sección se mostrarán las pruebas previas al paso a producción.

Sección 6: Discusión y Conclusiones: Se valorará el trabajo realizado y su comportamiento ya en un ambiente productivo, además se buscarán opciones de mejora como pasos hacia futuro.

2. Análisis

Para comenzar el proceso de análisis es buena práctica mapear el estado inicial de los procesos, en este caso es algo sencillo ya que todas las funcionalidades de negocio en las cuales se basa la orden de servicio tienen que ir y ser procesadas por el monolito existente en la base de datos que denominaremos CNET.

2.1. Diagrama C4



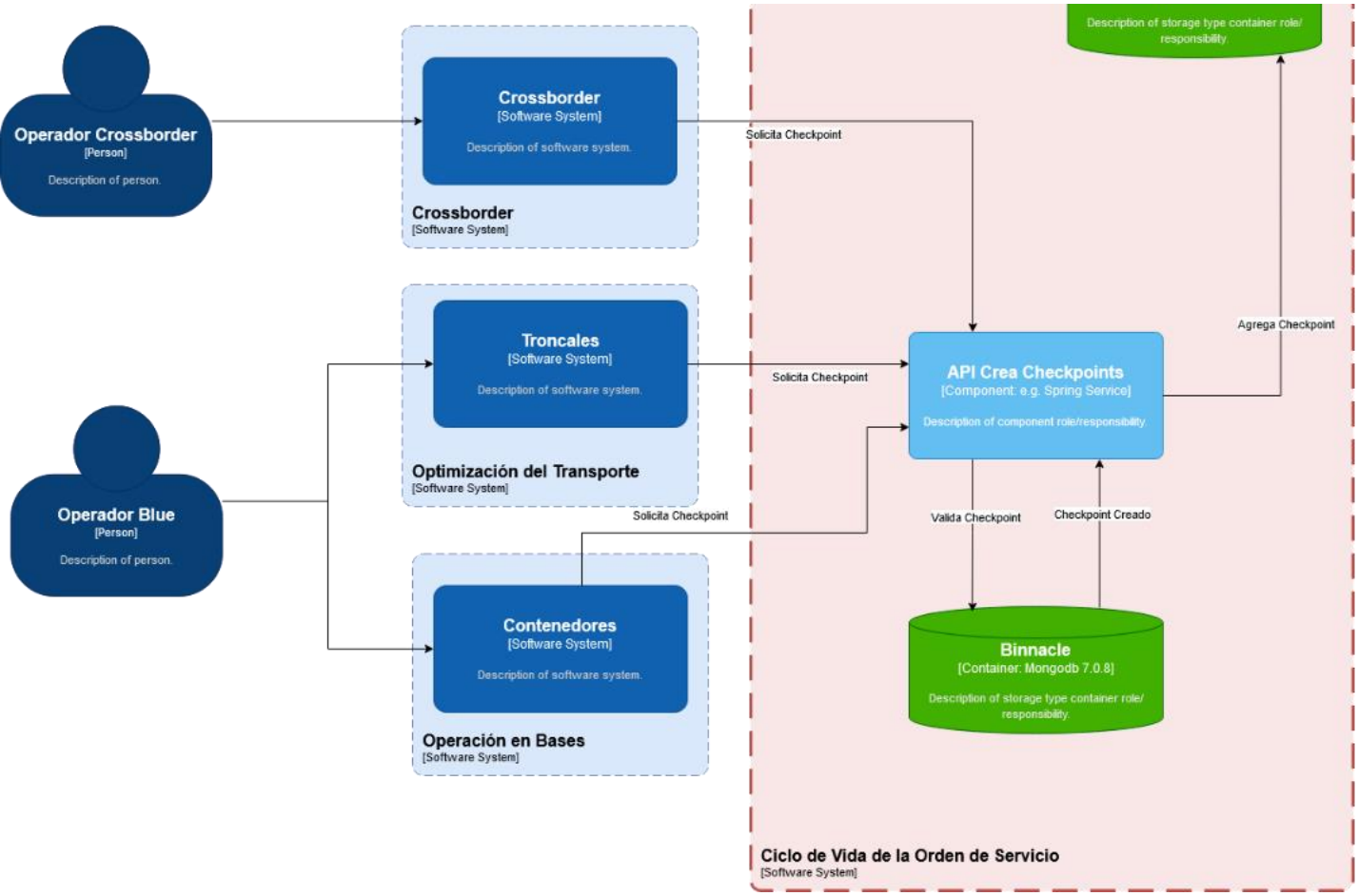


Figura 1: Diagrama C4 de nivel 1

El diagrama presente en la Figura 1 nos muestra una visión de alto nivel de como el sistema se relaciona con el universo de usuarios que hacen uso de este a través de sus sistemas de software, estos generalmente son los front-ends que utilizan los equipos para cosas como la emisión de ordenes de servicio. La actualización del seguimiento de los paquetes como es el caso de los troncales (consolidación de carga para diferentes zonas del país) y los contenedores (agrupación de paquetes pequeños dentro de un contenedor más grande). Anteriormente todos estos usuarios iban directamente a la base de datos operacional, en la búsqueda de una mejor robustez del sistema y mejor resistencia a fallas. Se propone de parte del equipo de Ciclo de Vida de la Orden de Servicio ir segregando el tráfico de los distintos usuarios a distintas entidades con tareas específicas, ya sean bases de datos o APIs, estos tres componentes que se observan en el diagrama, se ven en mayor profundidad en los diagramas que se presentan en la figura 2 y 3.

La figura 1 esta acotada solamente a la parte que se ve involucrada en este informe, pero esto no trabaja en un vacío hay diferentes procesos que el equipo también tiene como desafío desarrollar y desacoplar de la base de datos operacional.

El tener varios actores que interactúan en tiempo real con el sistema, gana relevancia el uso de la Arquitectura basada en eventos, ya que a través de Colas de mensajería y consumers/producers de mensajes, permite un procesamiento de tipo FIFO de los eventos que suceden con la carga o los mismos procesos operacionales y así mantener una lógica en los sucesos e información recopilada o entregada.

2.2. Tecnologías a Utilizar

Debido a las especificaciones de la empresa, las opciones de desarrollo se limitan a Java o Node para el Backend, como equipo se definió que para los casos generales las APIs se desarrollaran en NestJS y los consumers en Java, esto con la finalidad de mantener una homogeneidad en los repositorios que son creados y mantenidos por la célula de desarrollo.

En el caso específico de los servicios serán desarrollados en NestJS, además de eso se conectarán a una base de datos Mongo ya que una base de datos no relacional se ajusta de mejor manera a las necesidades del negocio y también simplifica la mantención de ésta en comparación a la complejidad agregada que traería una base de datos relacional. Más adelante se detalla en mayor profundidad sobre la estructura que se almacena en la base ya mencionada.

Previo a la implementación es obligatorio el generar la documentación referente a cualquier artefacto que se vaya a solicitar, en este caso para las APIs es necesario entregar un archivo que siga los lineamientos de Open API Specification (OAS por sus siglas en inglés) además de un diagrama de solución y uno de secuencia. Los cuales se encuentran adjuntos en el anexo.

Tal como se muestra en el diagrama, tanto para la API de Emissions Orders como para Get Order se utilizarán las siguientes tecnologías:

- **Node.js:**
  - **¿Qué es?:** Es un ambiente de ejecución para JavaScript que está basado en eventos y además de ello es asíncrono, enfocado en sistemas altamente escalables y por cómo está diseñado, a no ser que sea explícitamente escrito, las funciones que se pueden utilizar en Node.js no generan bloqueos. Uno de los focos de Node.js es la baja latencia y el uso de HTTP.
  - **Relevancia:** Básicamente es en donde se decidió que se ejecutarán los desarrollos para las APIs, se usa la versión 20.
- **NestJS:**
  - **¿Qué es?:** Es un Framework enfocado en el desarrollo eficiente y escalable de aplicaciones server-side en Node.js. Nest toma elementos de programación orientada a objetos, programación funcional y programación funcional reactiva, también soporta de manera nativa TypeScript y JavaScript, entre otras cosas también usa frameworks para HTTP como puede ser Express.
  - **Relevancia:** Es el framework específico a Node que se decidió para el desarrollo ya que también por cómo está pensado, hace más sencilla la adopción de los conceptos de Domain Driven Design

- **Jest:**
    - **¿Qué es?:** Jest es un framework en JavaScript usado para el testeo, enfocado en asegurar la correctitud de las codebases en JS. Permite a sus usuarios la escritura de tests con una estructura similar a la de otros frameworks de testeo, requiere de poca configuración y cuenta con reportes de cobertura.
    - **Relevancia:** Es el framework que se utiliza para hacer pruebas unitarias en los repositorios que usan TypeScript/JavaScript dentro de la célula de desarrollo.
- **OAS:**
    - **¿Qué es?:** Es un estándar propuesto por OpenAPI Initiative, el cual define una descripción agnóstica de los lenguajes de programación, la cual permite a computadoras y humanos, entender las capacidades de un servicio sin tener la necesidad de ver su código fuente, de esta manera se facilita el uso de APIs de parte de otros equipos o de nuevos desarrolladores.
    - **Relevancia:** Permite una comunicación consistente entre diferentes participantes del desarrollo.
- **JMeter:**
    - **¿Qué es?:** Es un software de código libre basado en Java diseñado para medir pruebas de carga funcional y registrar indicadores de performance.
    - **Relevancia:** Permite a los desarrolladores evaluar las necesidades técnicas de los servicios desarrollados, medir también la eficiencia de su código mediante las métricas que se obtienen a raíz de estas pruebas.
- **Datadog:**
    - **¿Qué es?:** Es un servicio de monitoreo para artefactos desplegados en cloud, el cual aporta métricas y logs de diferentes elementos, en el caso específico de BlueExpress se usa principalmente para ver el estado, salud y posibles mensajes de error que puedan salir en los servicios desplegados. Datadog también cuenta con alertas automatizadas configurables por el equipo de desarrollo.
    - **Relevancia:** Permite a los desarrolladores despejar errores, medir diferentes parámetros e incluso hacer un análisis con respecto a la performance de los diferentes aparatos de software.

2.3. Base de datos

Con el fin de guardar la información de las órdenes de servicio como se comentó anteriormente, se hace uso de una base de datos no relacional (NoSQL) proveída por MongoDB. El uso de esta tecnología se vio influenciado por su resiliencia ante futuros cambios, además de que se adapta mejor a las necesidades que presentan las ordenes de servicio, las cuales son más bien de un tipo semiestructurado, con el uso de colecciones y documentos esto se logra satisfacer de buena manera.

En este documento se encuentran datos que sirven para individualizar la orden de servicio, a la persona o empresa quien envía el producto y su dirección, la fecha de creación de la orden, quien es el destinatario y los datos de su dirección, también se almacena información referente a las medidas, peso del paquete y el seguimiento de este.

Esto nace de la necesidad de mantener una trazabilidad del camino de la carga y las responsabilidades correspondientes en caso de algún problema durante el proceso.

Aquí se detalla a modo de ejemplo el cómo se conforma un documento con la información que podría tener una orden de servicio:


```
{
  "_id": {
    "$oid": "656e35042aa78b5e2c7be2df"
  },
  "orderId": "8011124656",
  "emissionDate": "2023-12-04T00:00:00Z",
  "seller": {
    "account": "77088208-1-8",
    "name": "Compañia de prueba SA",
    "identifier": "19.523.090-0",
    "fullAddress": "General Flores 60, PROVIDENCIA, Región Metropolitana"
  },
  "buyer": {
    "name": "camila sepu",
    "fullAddress": "General Flores 2267, PROVIDENCIA, Región Metropolitan"
  },
  "packages": [
    {
      "packageId": "1317776316",
      "weight": 4,
      "height": 0,
      "width": 0,
      "length": 0,
      "units": {
        "weight": "KG",
```

```
        "length": "CM"
      },
      "referenceChildOrder": "8011124656",
      "trackings": [
        {
          "trackingId": 288,
          "eventDate": {
            "$date": "2024-05-07T13:42:47.100Z"
          },
          "eventCode": "GE",
          "eventType": "EX",
          "location": "DZOS-1-ZAL2C",
          "creationDate": {
            "$date": "2024-05-07T13:42:47.100Z"
          },
        }
      ]
    }
  ],
}
```


2.4. Historias de Usuario

a. Como Fulfillment

- **Quiero** obtener toda la información de la Orden de Servicio, consultando por el número de seguimiento.
- **Para** desplegar la información de la orden de servicio al usuario en el portal.

Criterios de Aceptación

- La información de la orden de servicio debe estar disponible en menos de 3 segundos.
- En caso de no existir información disponible según el número de seguimiento consultado, se debe devolver mensaje explicativo y código de error de negocio.
- Debe considerar la lectura de los dos tipos de códigos de barras que utiliza actualmente la compañía.

b. Como Cliente

- **Quiero** crear Orden de Servicio en el nuevo modelo de “Orders”.
- **Para** iniciar el proceso de envío de paquetes desde una plataforma estable y rápida.

Criterios de Aceptación

- El proceso debe tardar menos de 5 segundos.
- El servicio debe retornar el estado final del proceso con un OK o NOK.
- El Servicio debe retornar el número de seguimiento asignado a la nueva orden.

2.5. Descripción de la Propuesta

Al momento de llegar al equipo están hechos los consumidores de **orders-inyected** y **orders-modified** que se alimentan de su respectivo **tópico** y envían mensajes a su **cola**, lo que hacen estos es generar mensajes con la información de las órdenes de servicio que vayan a ser creadas o modificadas sobre la nueva base de datos no relacional (MongoDB) la cual también va a estar de manera progresiva trayendo data que se encuentre en la antigua base de datos.

Para empezar, los esfuerzos de desarrollo se enfocan en el dominio de las órdenes de servicio, esto debido a que la primera prioridad a migrar es el proceso de emisión de órdenes de servicio y con ello también la consulta de estas.

Es necesario que la información de las órdenes pueda ser obtenida limpiamente e independiente del origen para cualquier usuario o equipo que la requiera, esto se hará a través de un microservicio con un componente de autenticación. Con eso tenemos el requerimiento para la primera API a implementar la cual está enfocada en la obtención de la información de las órdenes de servicio.

La segunda API a crear es la que cumple con la función de poder crear órdenes de servicio en el nuevo modelo.

La tercera API es llamada por nuestra segunda API (la de creación de órdenes) para poder obtener el número de seguimiento de la orden de servicio. Esta API es “temporal” y se usará durante el tiempo de la migración ya que su función es ir a buscar el número a la base del legado ya que el otorgamiento de este identificador cumple con cierta lógica que se encuentra en CNET, en una etapa posterior este proceso será replicado en el nuevo modelo con la lógica correspondiente.

Finalmente nos cambiamos al dominio de tracking operacional el cual nos permite aislar y migrar la lógica referente a lo que en la empresa se conocen como pinchazos, formalmente los pinchazos son cada punto de control por el que pasa un paquete, dígase, cuando la carga es recibida por la empresa, pasa al centro de distribución, la toma un courier o se le entrega al cliente. Esos son algunos casos y eventos necesarios a registrar y relacionar a una orden de servicio.

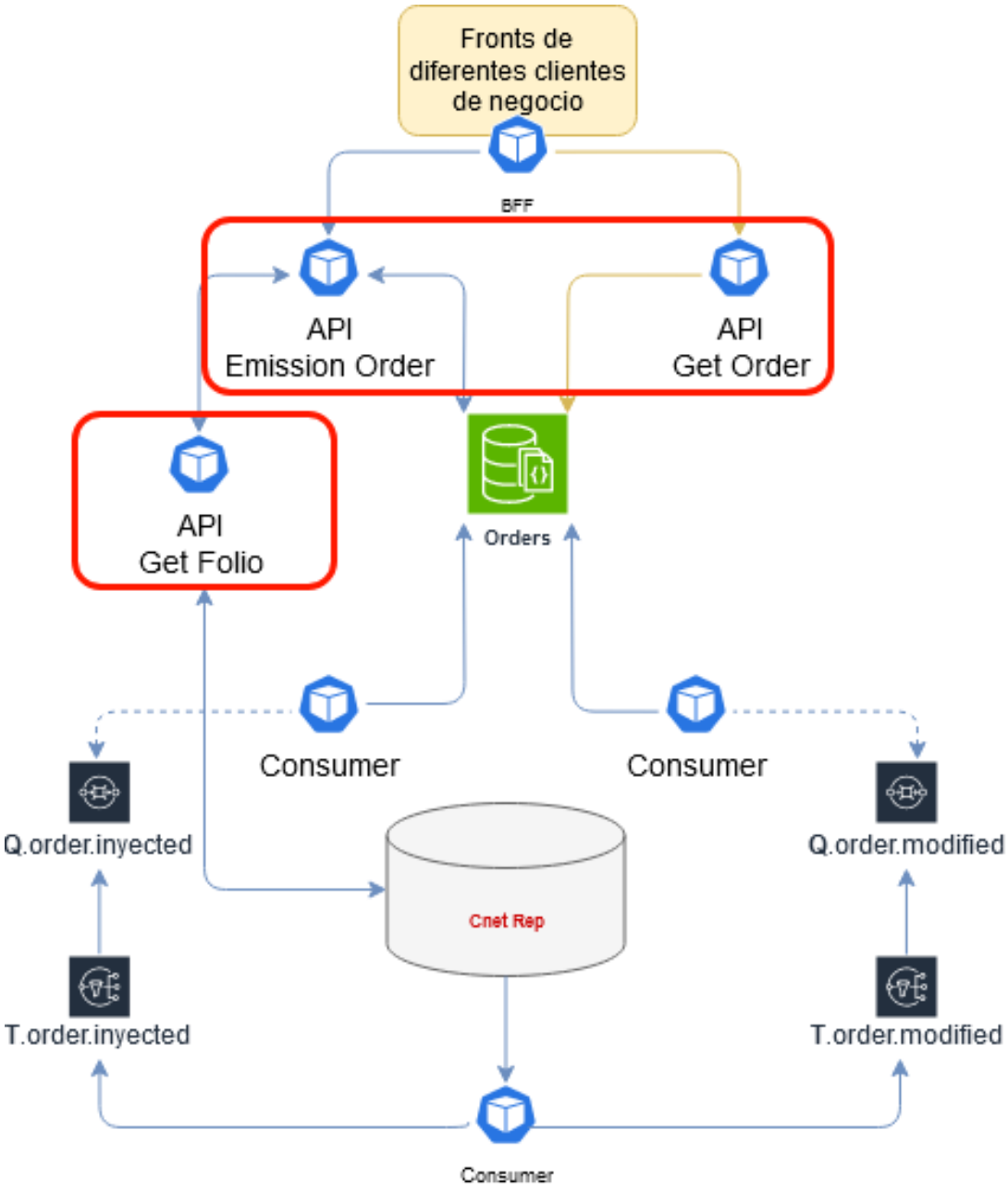


Figura 2: Diagrama del dominio de ordenes de servicio

La solución que se encuentra en la Figura 2, es parte del flujo de negocio que cumple con la finalidad de extraer un proceso que solía hacerse íntegramente en la BBDD de Oracle, como puede ser la creación y consulta de órdenes, ahora se logra desacoplar del viejo modelo, llevándolo a una arquitectura más flexible y escalable. En la mitad inferior del diagrama está el flujo que se responsabiliza de llevar la información existente desde la base de datos Oracle a la cual de manera interna se le denomina Cnet hacia el nuevo modelo a través de mensajes hasta la base de datos, para ello pasa por los consumidores que ya estaban creados por el equipo, el order-inyected y el order-modified.

En la otra mitad superior del diagrama vemos cómo al tener un usuario que interactúe con un front-end que consuma cualquiera de nuestras APIs, estas van a leer o escribir información directamente en la base de MongoDB y específicamente en el caso de la emisión será necesario ir a buscar el número de seguimiento a CNET mediante la API de Get Folio.

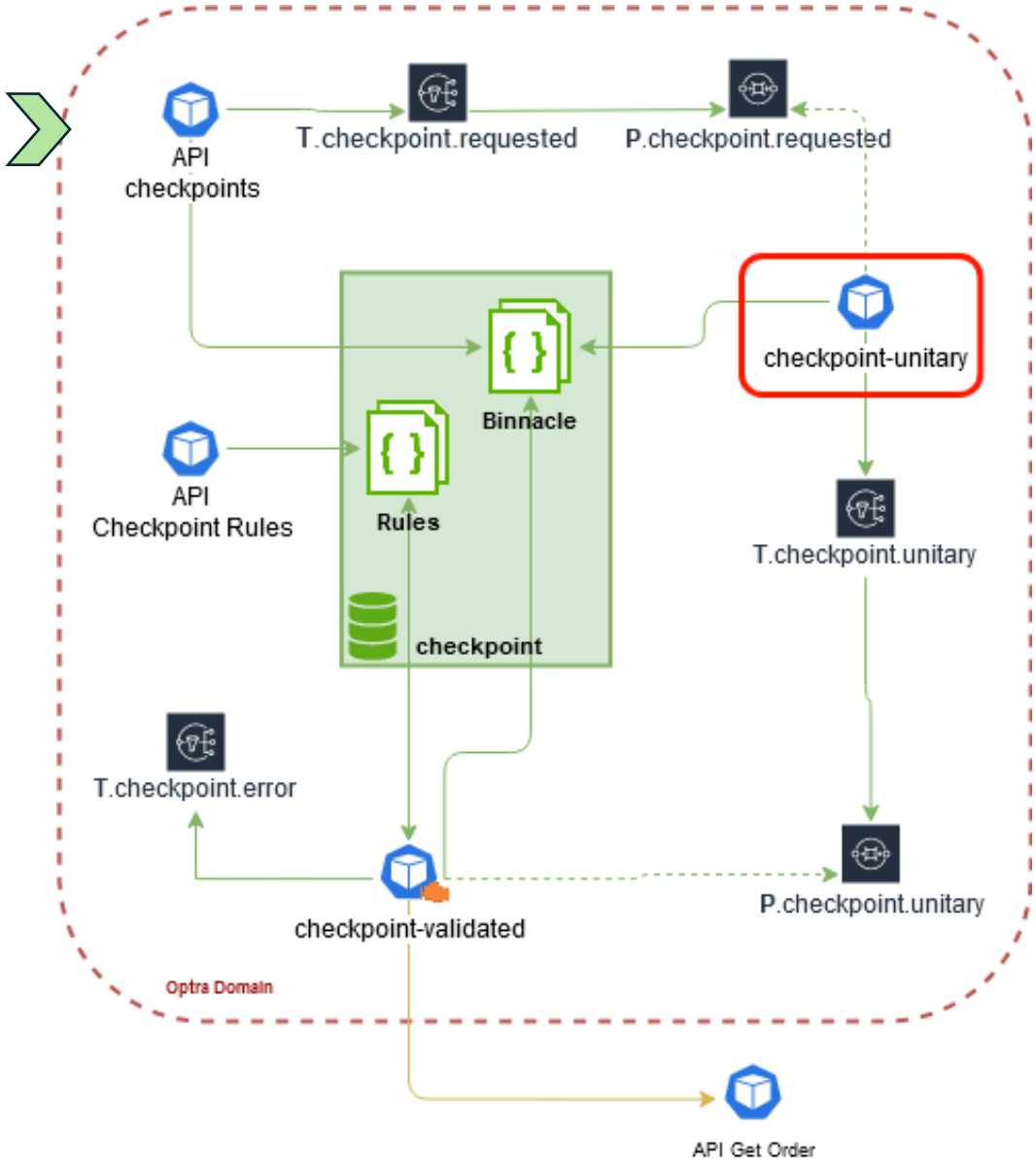


Figura 3: Diagrama de solución para el dominio de tracking operacional

En la Figura 3, se muestra el flujo de los pinchazos, este se compone de tópicos, queue, algunos consumers y servicios, los pinchazos son una estructura de información que se encarga de mostrar donde se encuentra la carga que se está transportando y sobre quien recae la responsabilidad de esta, los pinchazos pueden venir de manera masiva o unitaria en una request a la API checkpoints, posteriormente el consumer a desarrollar llamado checkpoint-

unitary tiene la tarea de desagregar estos pinchazos y enviar mensajes con pinchazos de manera individual a la colección llamada Binnacle que es la bitácora de los ellos y posteriormente al tópico correspondiente para que el motor de reglas que se encuentra en checkpoint-validated pueda verificar de que estos pinchazos cumplan con las reglas de negocio necesarias y descarte los pinchazos que no las cumplan.

2.6. Requerimientos

Estos artefactos deben cumplir con las definiciones de calidad definidas por arquitectura además de las impuestas por el equipo. Este tipo de condiciones son:

- Cumplir por lo menos hasta el segundo nivel del modelo de madurez de Richardson [5], esto implica usar correctamente los verbos HTTP y asignar sustantivos para los recursos en la URL.
- Contar con un 80% de cobertura en tests unitarios.
- Se debe contemplar en el diseño el patrón EDA [6]
- La estructura del repositorio debe ser ad hoc al Domain Driven Design [7]

3. Diseño y Documentación.

La documentación para la API de consultas es la siguiente:

- Diagrama de Solución

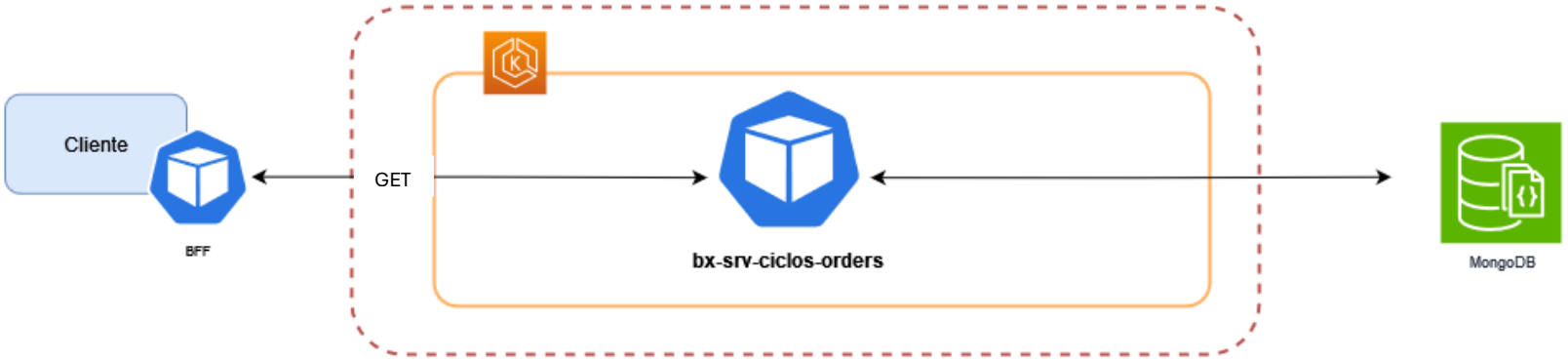


Figura 4: Diagrama de solución para la API de consultas sobre órdenes.

En la Figura 4 se muestra como la API interactúa con su flujo, entonces un cliente/usuario desde un front-end que tiene un BFF (Back-end For Front-end) le hará un request GET a la api de nombre bx-srv-ciclos-orders la cual busca información en la MongoDB para posteriormente devolvérsela al cliente.

- Diagrama de Secuencia

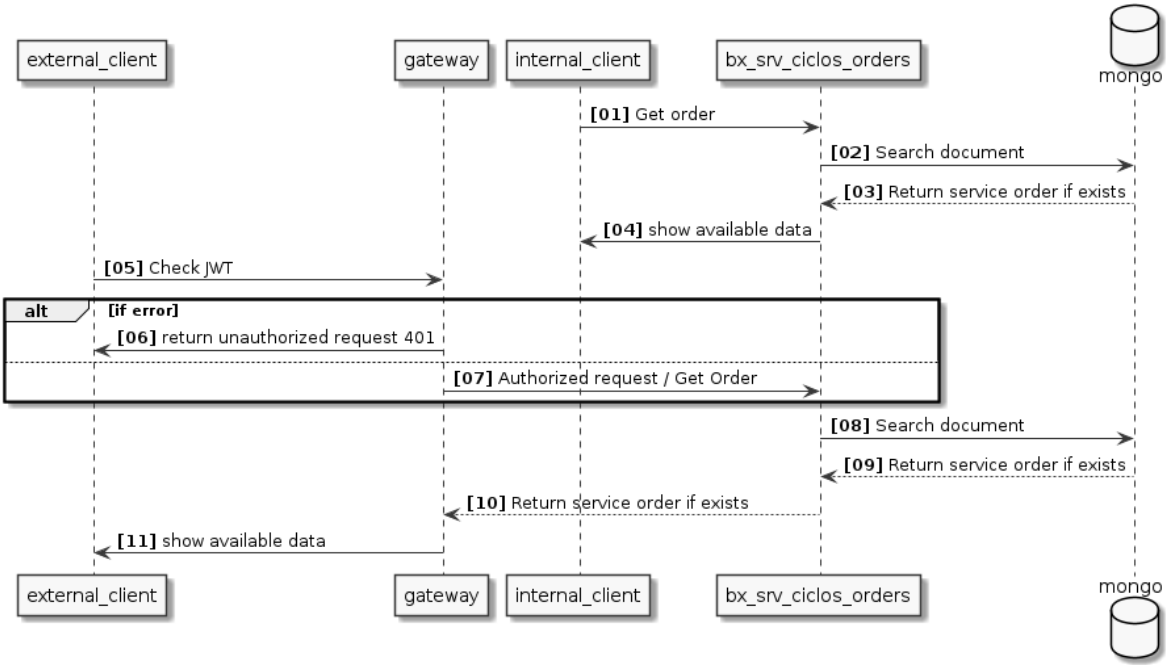


Figura 5: Diagrama de secuencia para la API de consultas sobre órdenes.

En la Figura 5 vemos como es el camino que recorre una request, sea esta desde un cliente externo o uno interno. Para el primer caso, primero se hace la validación de autenticación a mediante JWT a través de un API Gateway en este caso proveído por Kong[8], en caso de no ser una autenticación válida se devuelve un error 401, en caso de que la autenticación sea correcta, el funcionamiento es igual que para el cliente interno, se ejecuta el comando hacia la API de Get Order, la API busca el documento correspondiente y devuelve la información necesaria, posteriormente el servicio devuelve una respuesta tal como se especifica en el que se encuentra en el Anexo 1 al final de este documento, no obstante aquí se encuentra una respuesta de ejemplo con la información más relevante.

- Respuesta de ejemplo:


```
{
  "orderId": "3000470255",
  "emissionDate": "2024-05-15T15:30:00.000Z",
  "quantityPackages": 1,
  "seller": {
      "name": "Seller Company",
      "identifier": "96787360-8"
  },
  "buyer": {
      "name": "John Dwayne"
  },

  "shipper": {
      "account": "76273299-1-8"
  },
  "pickupAddress": {
      "street": "123 Main St",
  },
  "deliveryAddress": {
      "street": "456 Side St",
      "communeDesc": "IQUIQUE",
      "regionDesc": "Tarapacá",
      "fullAddress": "456 Side St 789, IQUIQUE, Tarapacá",
  },
  "packages": [
      {
          "packageId": "3000470266001",
          "inspection": {
              "weight": 12,
              "height": 23,
              "width": 34,
              "length": 34,
              "units": {
                  "weight": "KG",
                  "length": "CM"
              }
          }
      }
  ]
}
```

La documentación para la API de creación de órdenes de servicio es la siguiente:

- Diagrama de Solución

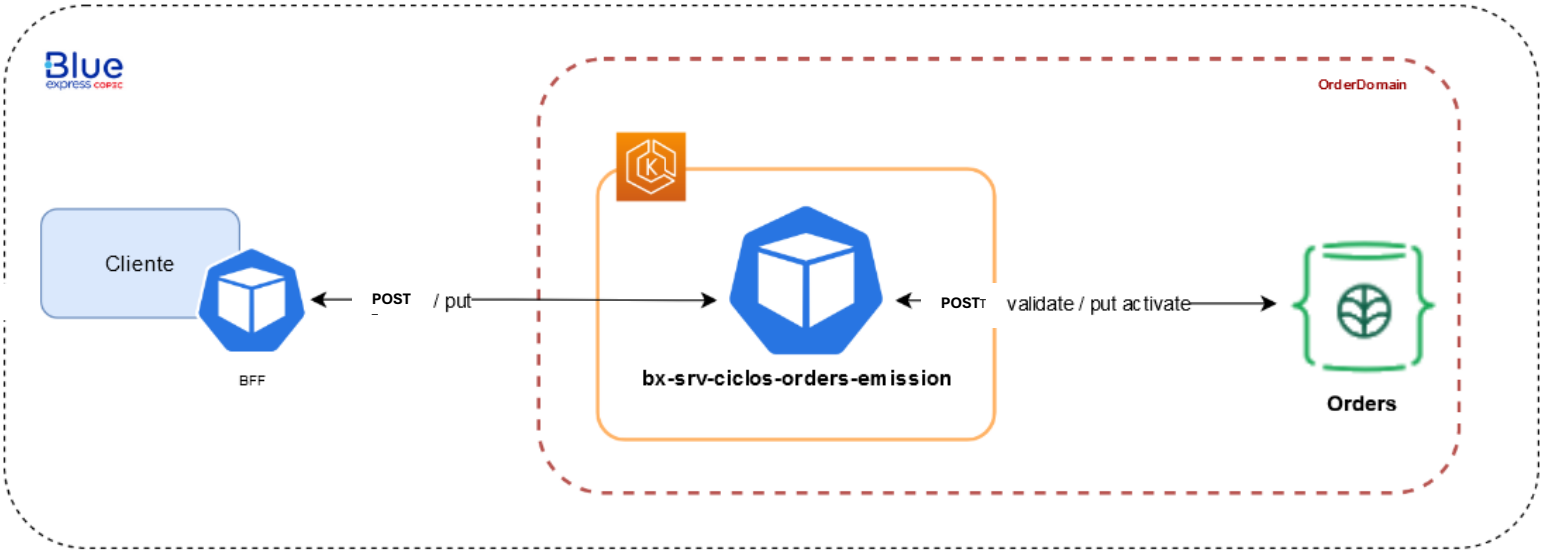


Figura 6: Diagrama de secuencia para la API de creación de órdenes de servicio.

En la Figura 6 se muestra como la API interactúa con su flujo, entonces un cliente/usuario desde un front-end que tiene un BFF (Back-end For Front-end) le hará un request POST o PUT a la api de nombre bx-srv-ciclos-orders-emission la cual busca generar, activar o validar según corresponda hacia la MongoDB para posteriormente devolver información de la operación realizada al cliente, principalmente dando a conocer el número de orden que se le asignó al cliente.

- Diagrama de Secuencia

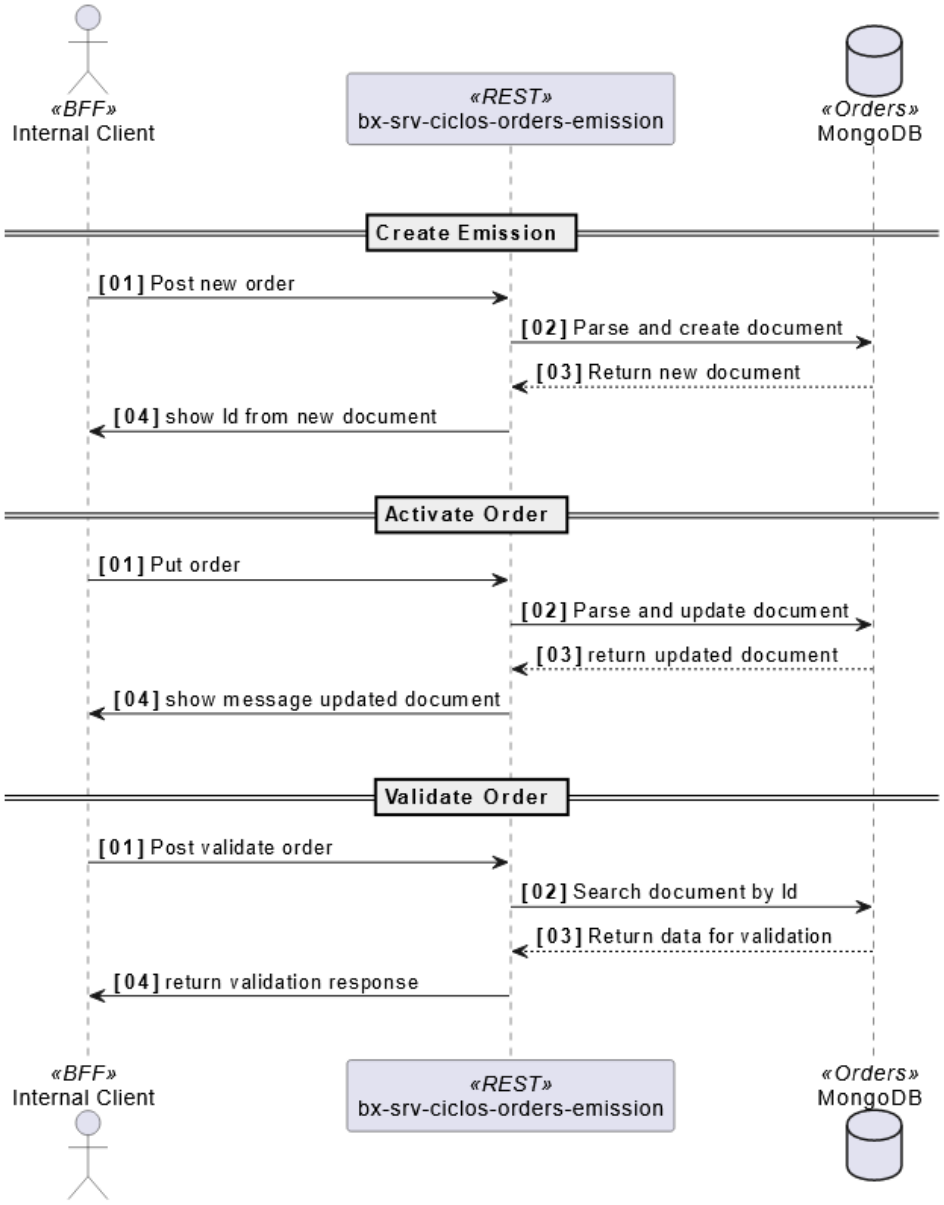


Figura 7: Diagrama de secuencia para la API de emisión de órdenes.

En la Figura 7 vemos como es el camino que recorre una request, a diferencia de la API anterior, esta no es consumida por clientes externos, por consiguiente no es necesaria la validación a través de JWT ya que la instancia se encuentra dentro de la red interna de AWS.

Vemos que en el caso de Creación, se toma la información y se crea el documento correspondiente y devuelve el Id del nuevo documento, en el caso de activar la orden, se retorna un mensaje diciendo que el proceso fue exitoso y para las validaciones devolvemos si la validación fue exitosa o no.

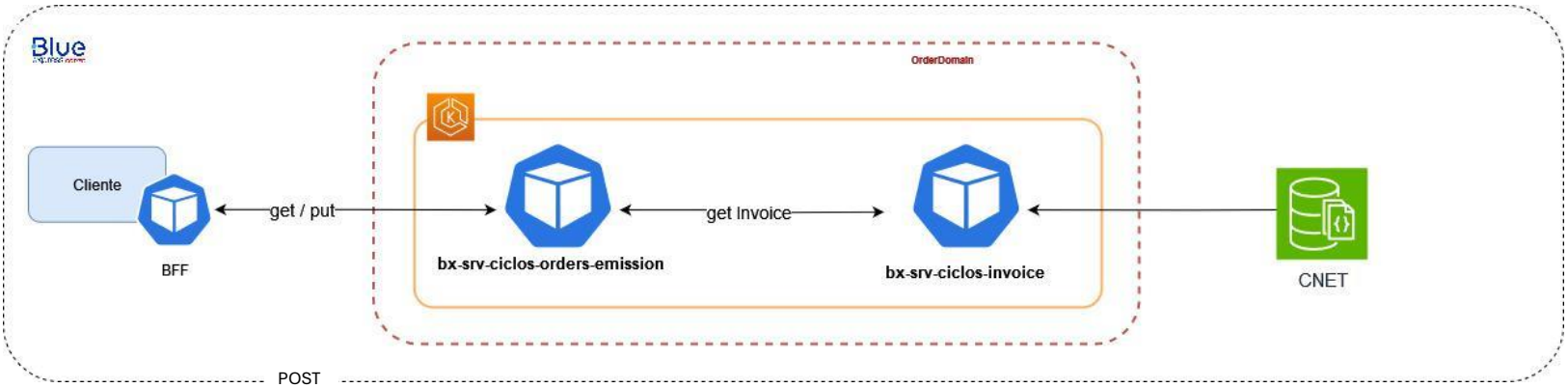


Figura 8: Diagrama de solución para el Servicio de obtención de folios

En la Figura 8 se muestra como la API interactúa con su flujo, cuando un cliente/usuario desde un front-end que cuenta con su backend respectivo (Back-end For Front-end) y este le hace un request POST a la API de nombre bx-srv-ciclos-orders-emission, esta efectuará un request GET a bx-srv-ciclos-invoice para que esta pueda obtener un folio desde la base de datos CNET.

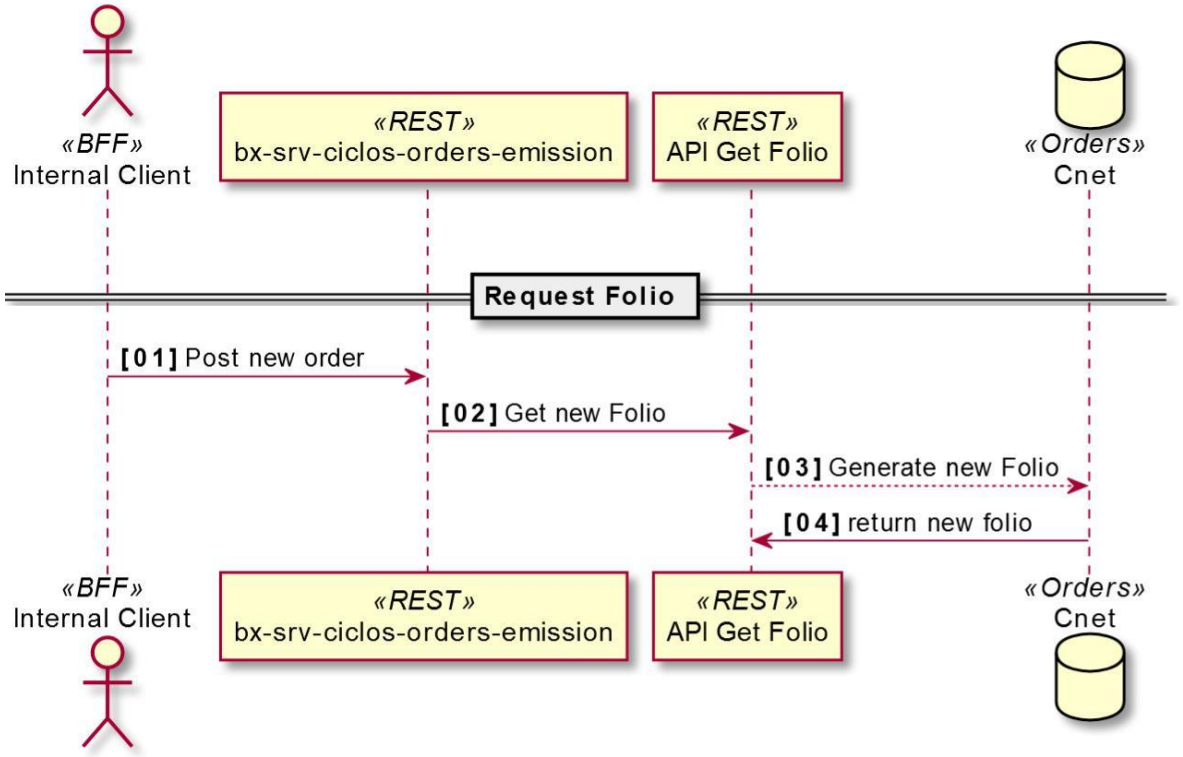


Figura 9: Diagrama de secuencia para el Servicio de obtención de folios

En la Figura 9 vemos como es el camino que recorre una request, en este caso, esta también viene desde un cliente interno, el cual le ejecutará una request a orders-emission posteriormente se le hace el request de Get new Folio a la API de Invoice y finalmente se devuelve el número de seguimiento que devuelve CNET desde su stored procedure.

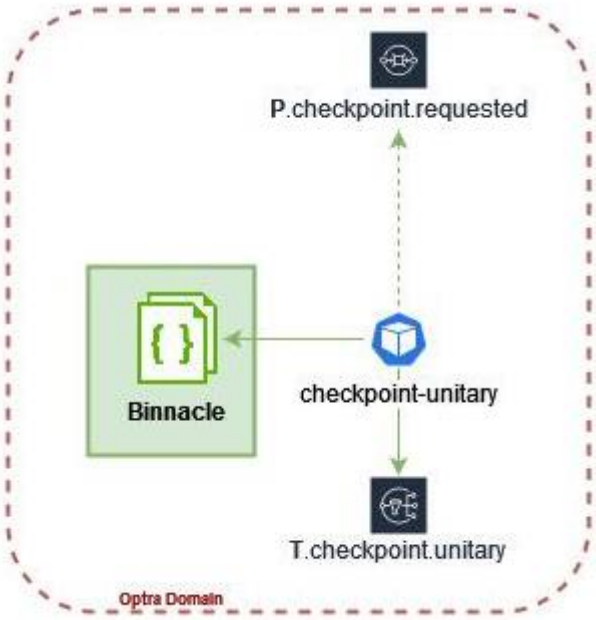


Figura 10: Diagrama de solución para el Consumer que desagrega pinchazos.

En la Figura 10 se muestra como el consumer participa del flujo, el consumer recibirá los mensajes desde la Queue Checkpoint.requested, los procesará para desagregarlos, modificará el estado del pinchazo en la collection de MongoDB llamada Binnacle y finalmente publicará un mensaje al tópico de SNS checkpoint.Unitary, tal como lo podemos ver el paso a paso en la Figura 11.

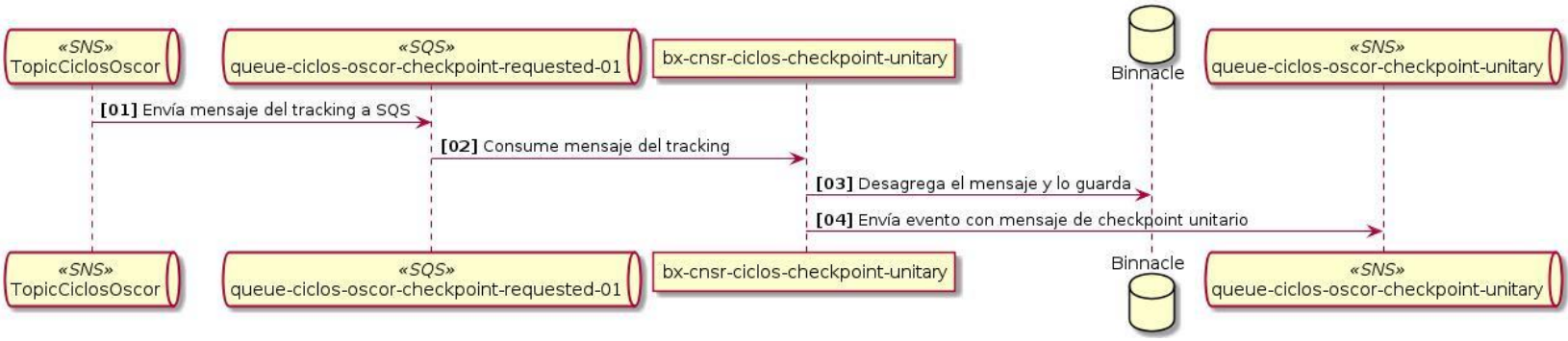


Figura 11: Diagrama de secuencia para el Consumer que desagrega pinchazos.

4. Evaluación

4.1. Pruebas Unitarias

Esto es una técnica fundamental al momento de hacer software de calidad, nos permite probar y asegurar el funcionamiento de artefactos de software de tal manera que los componentes y sus funciones de manera individualizada puedan ser analizados y probados para la pronta detección de cambios en el comportamiento, ya sea por modificaciones al servicio o por algún error de programación, esto ayuda ya que se valida antes de que los cambios sean desplegados en cualquier ambiente, estas pruebas son ejecutadas y si no son superadas con éxito, el flujo se interrumpe y se evitan problemas mayores.

En este caso en es Jest la herramienta que se utiliza para escribir las pruebas unitaria, este framework de pruebas también nos permite el cálculo de cobertura de las mismas, por consiguiente, se van generando pruebas para las funciones que se encuentran en las APIs hasta alcanzar una cobertura mayor o igual al 80%.

Para poder hacer pruebas unitarias, nos enfocaremos en probar elementos individuales de nuestro software, aislando cualquier elemento externo a través de los denominados “Mocks”

Esto se ve al comienzo de nuestro archivo de pruebas, posteriormente por requerimiento de Jest es necesario describir la clase que probamos y describir el método.

Una vez estamos describiendo el método es donde comienza lo que se denomina “Arrange, Act and Assert”

Donde por cada etapa hay cosas características, por ejemplo, en el Arrange se prepara toda la data sintética para el test.

En el Act se llama la función a probar en este test.

Finalmente, en el Assert hacemos las validaciones del comportamiento que esperamos para esa prueba en específico.

Un archivo de pruebas a una clase puede tener varios tests.

A continuación, se muestra como es un archivo de pruebas unitarias resumido a un solo test para no extender tanto el ejemplo.

```
const mockFindOne = jest.fn();
const mockCollection = {
  findOne: mockFindOne
};

const mockDb = {
  collection: jest.fn().mockReturnValue(mockCollection),
};

describe('ConfigurationRepository', () => {
  let configurationRepository: IConfigurationRepository;
  describe('getConfigurationByAccount', () => {
    it('should retrieve configuration by accountId from the database', async
() => {
//ARRANGE
      const mockData = {
        accountId: '12312319-12-1',
      };

      mockFindOne.mockResolvedValue(mockData);

      const account = Account.fromPlain('12312319-12-1');
//ACT
      const result: Configuration =
        await configurationRepository.getConfigurationByAccount(account);
//ASSERT
      expect(mockCollection.findOne).toHaveBeenCalledWith({
        accountId: account.toPlainString(),
        state: 'V',
      });
      expect(result).toBeInstanceOf(Configuration);
    });

});
});
```

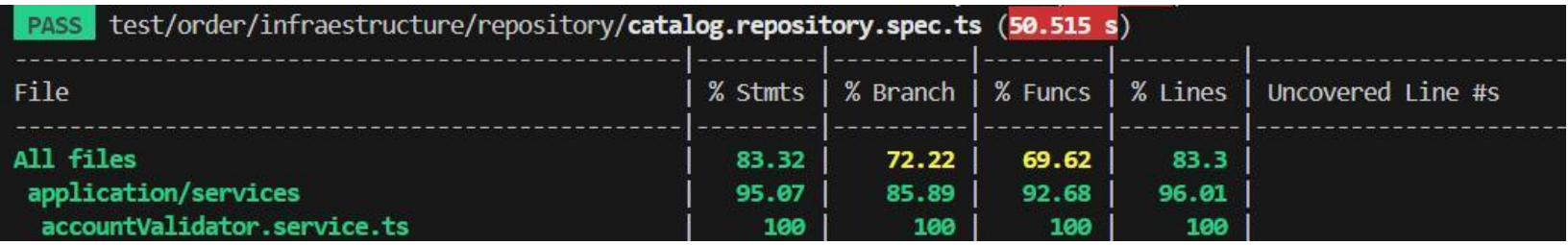


Figura 13: Output de cobertura posterior a la ejecución de Jest.

Una vez que ya se programan las pruebas unitarias, es necesario ejecutarlas para calcular cuánto porcentaje del código ha sido probado y cubierto de manera exitosa, en el caso del repositorio que se está probando, se ha logrado un 83% de cobertura tal como se puede apreciar en la Figura 13 y con esto se cumplen las definiciones impuestas por el departamento de arquitectura.

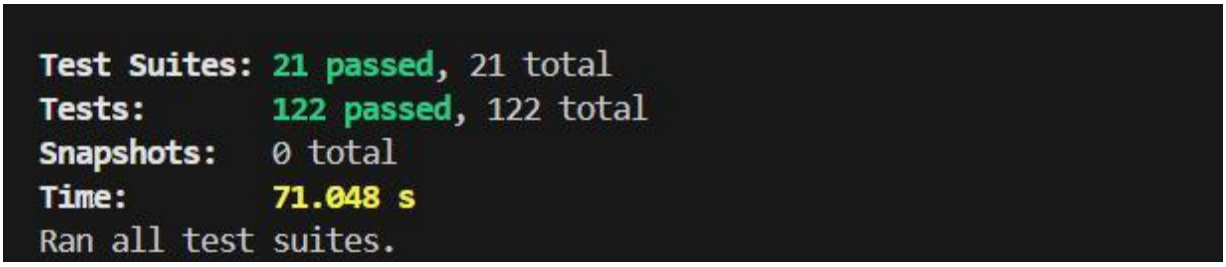


Figura 14: Output general de la ejecución de pruebas en Jest.

En este repositorio en concreto y como observamos en la Figura 14 ya hay programados más de 120 pruebas para un total de 21 archivos.

4.2. Pruebas de Estrés

Esta parte consiste en enviarle tráfico sintético al servicio con tal de exigirle un caso límite y ver como es el consumo de CPU de su instancia en cloud, con la finalidad de hacer ajustes tanto esta misma cómo buscar posibilidades de optimización de código, por estándar de la empresa para las APIs son como mínimo 1000 requests en 60 segundos, pero si se sabe que el servicio tendrá una mayor demanda también se pueden hacer pruebas más exigentes, al momento de finalizar la ejecución se espera que la API no sobrepase el 80% de uso de CPU, con tal de evitar inestabilidad en caso de carga, ya que al sobrepasar el máximo la instancia dejará de responder y se reiniciará teniendo como consecuencia un downtime no programado.

A continuación, se muestra la prueba de estrés que se realizó a uno de los servicios previo al paso a producción.

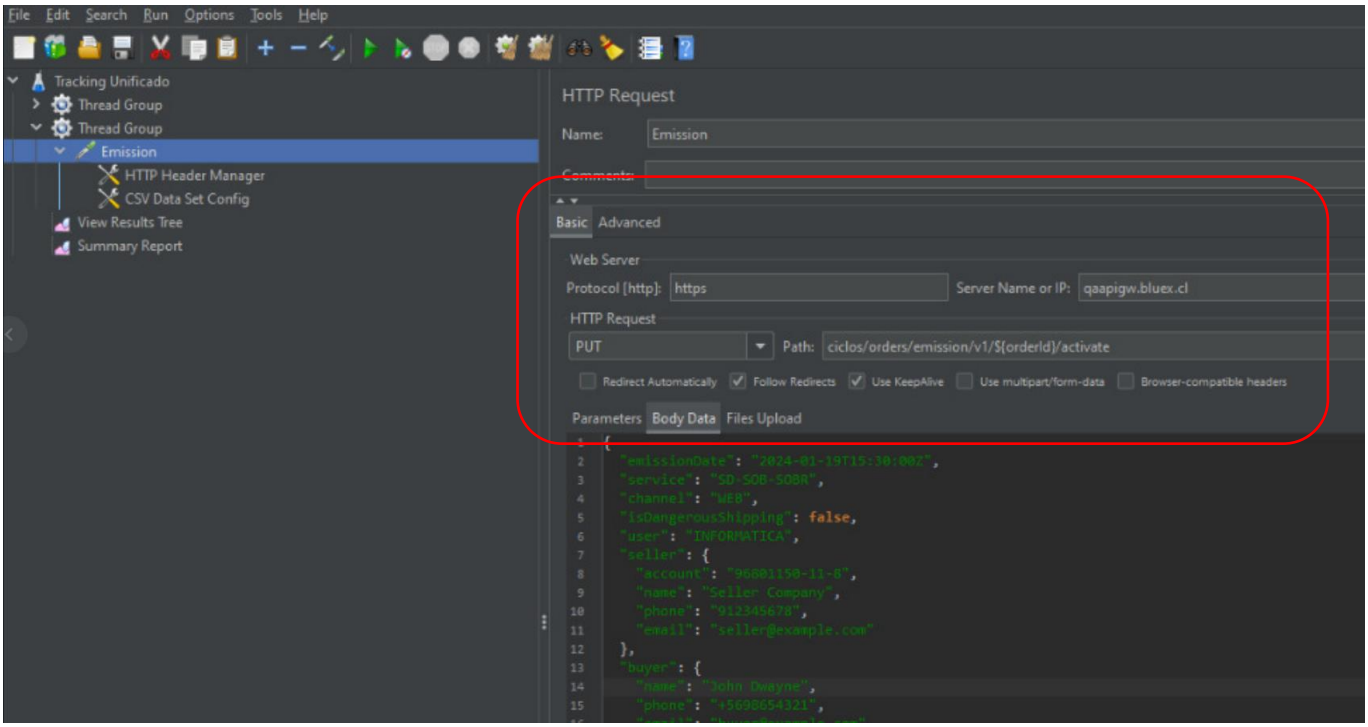


Figura 15: Se setea la url y el body del request para la generación del tráfico sintético.

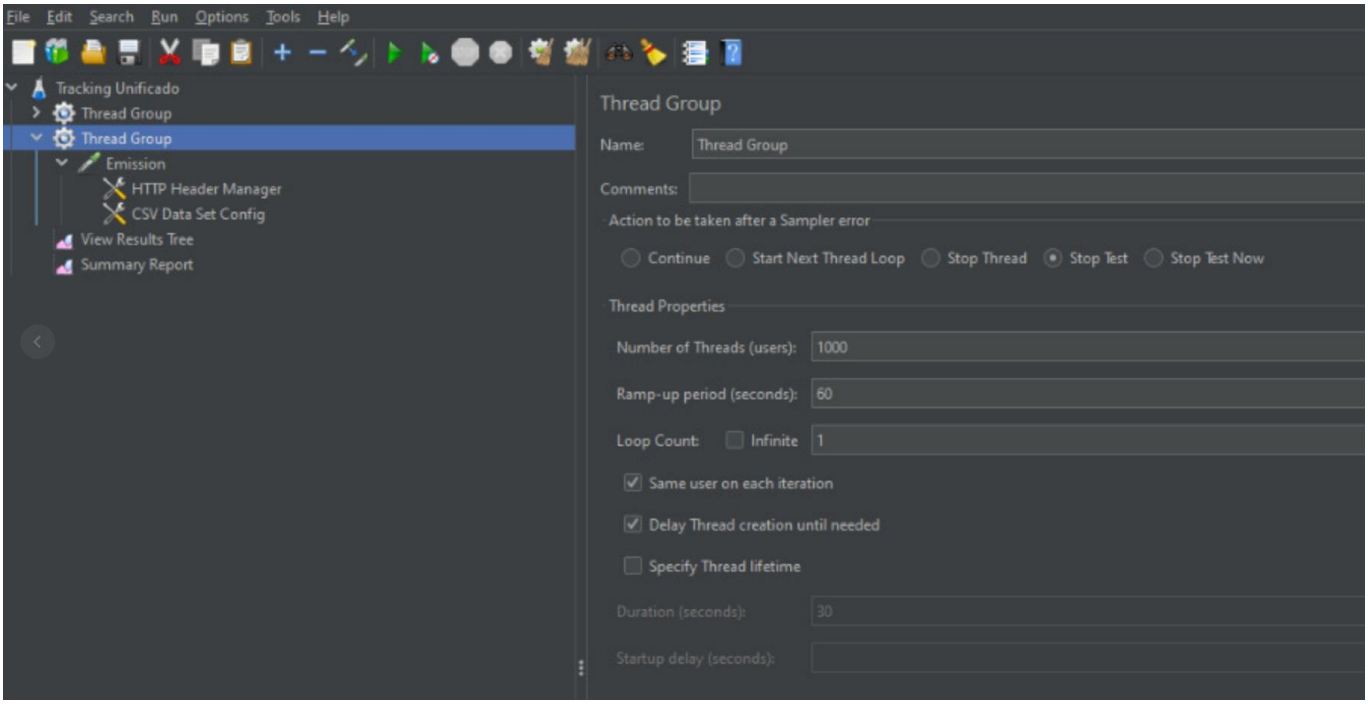


Figura 16: Se configura la cantidad de requests que se ejecutarán y también el tiempo.

El mínimo a probar son 1000 requests en 60 segundos, por consiguiente se prueba con ello y en caso de ser necesario se ajustan los valores de RAM y CPU de las instancias.

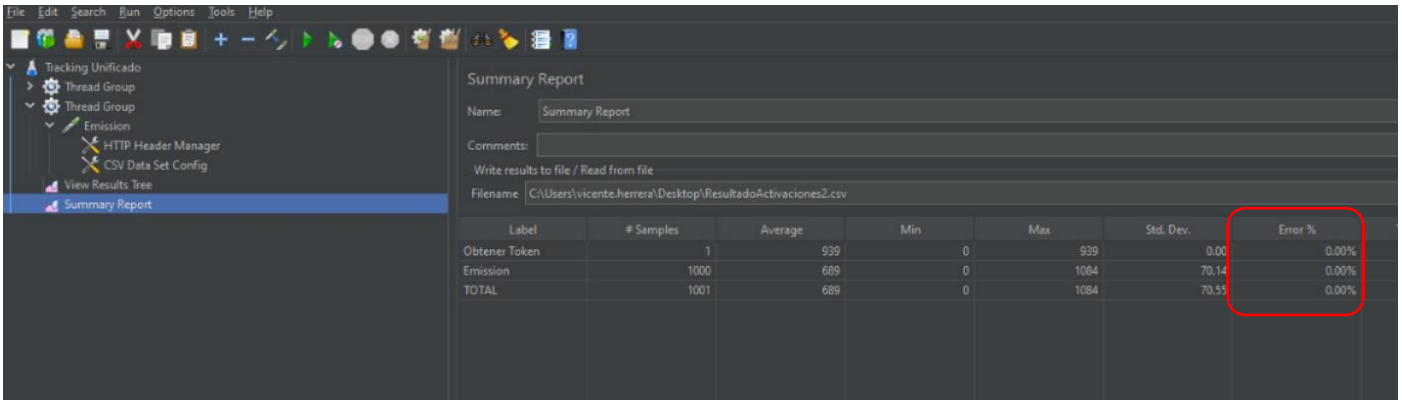


Figura 17: Este panel muestra las estadísticas de la prueba.

Al momento de revisar este reporte es importante que el porcentaje de error sea 0%, en caso de que no sea 0% es necesario revisar si los errores son por razones del request o del servidor y viendo los mensajes que devuelve la API.

Figura 18: Se utiliza Datadog para revisar el uso de recursos.

Se observa un peak de 17.08% de uso de CPU y un uso de memoria normal, por consiguiente, no es necesario efectuar ajustes al hardware de la instancia.

5. Discusión y Conclusiones

Con los nuevos servicios en producción se puede ver como mejora la performance, la fiabilidad y la escalabilidad de estos sistemas.

Es importante destacar que con este informe no se cierra a las arquitecturas monolíticas, ya que existen matices, donde estas claramente funcionan y cumplen a cabalidad con las necesidades tanto de negocio como de complejidad. No es extraño pensar que para sistemas más pequeños sean una alternativa viable. Pero también es importante mostrar que el paso de una arquitectura monolítica a una distribuida, de forma gradual y progresiva es posible, además de no ser un cambio disruptivo, sino que se puede buscar que sea consensuado y gradual para no afectar al resto del negocio.

También cabe destacar que la nueva arquitectura aún no se encuentra finalizada pero con las implementaciones ya hechas, existe una diferencia en lo que se buscaba mejorar, esto ha sido positivo ya que se cumplen los objetivos de mejorar la fiabilidad ante posibles eventualidades o anomalías en la carga del sistema como podría ser un “CyberDay” por ejemplo.

Con esto se mejora al final del día la experiencia del usuario ya que se encuentra frente a un sistema más robusto en todas sus aristas, teniendo menos probabilidades de encontrarse una web o un sistema que no funcione, además permite mayor flexibilidad desde la parte de ingeniería o de negocio para buscar e implementar nuevas funcionalidades que por limitaciones o restricciones de la antigua arquitectura eran simplemente imposibles de llevar a cabo.

Recapitulando sobre los objetivos inicialmente planteados también vemos que se logró satisfactoriamente el estudio de cómo funciona la compañía, también se diseñó, documentó, desarrolló y probó el nuevo software requerido para completar las necesidades de los flujos que se migraron.

Finalmente, el desarrollo de esta migración continuará con desacoplar el proceso de facturación completo y el de tarificación, cada uno de estos cuenta con una arquitectura similar dígase APIs, consumers y bases de datos.

6. Referencias

[1] Blue Express. Nosotros. Ultima consulta el 11 de mayo de 2024 en **https://www.blue.cl/nosotros/0**

[2] Cámara de Comercio de Santiago. Estudio 2021. Ultima consulta el 11 de mayo de 2024 en **https://www.ecommerceccs.cl/wp-content/uploads/2021/10/glever-SUMMIT2021-short2.pdf**

[3] **https://docs.aws.amazon.com/es_es/sns/latest/dg/welcome.html** . Ultima consulta el 12 de agosto de 2024

[4] Scrum Guide by Ken Schwaber and Jeff Sutherland **https://www.scrum.org/resources/scrum-guide**

[5] *Kanban - Successful Evolutionary Change for your Technology Business*. Anderson, David (abril de 2010)

[6] *Richardson, Leonard (2008-11-20).* https://www.crummy.com/writing/speaking/2008-QCon/act3.html

[7] Event-Driven Applications: Costs, Benefits and Design Approaches

https://web.archive.org/web/20131023055205/http://infospheres.caltech.edu/sites/default/files/Event-Driven%20Applications%20-%20Costs,%20Benefits%20and%20Design%20Approaches.pdf

[8] Evans, E. (2004). *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley.

[9] https://konghq.com/products/kong-gateway

7. Anexos

Anexo 1: OpenAPI Specification de API Get Order:

```yaml
openapi: 3.0.3
info:
  title: API Service Orders
  description: |-
    API that displays read-only data from the Service Order
  version: 1.0.0
  contact:
    name: Developer Team
    email: vicente.herrera@blue.cl
servers:
  - url: http://localhost:8080
    description: Local Enviroment
tags:
  - name: Service Orders
    externalDocs:
      description: Find out more
      url: http://swagger.io
paths:
  /api/ciclos/orders/v1/{orderId}:
    get:
      tags:
        - Service Orders
      summary: Finds Service Orders by orderId
      description: If request is successful the data from the service order
will be displayed
      operationId: readOnlyGet
      parameters:
        - name: orderId
          schema:
            type: string
          in: path
          required: true
      responses:
        '200':
          description: Resource obtained successfully
          content:
            application/json:
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Order'
        '400':
          description: An error occurred while geting the resource
```

```yaml
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'
        '404':
          description: The resource you were trying to reach cannot be found
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/Error'

components:
  schemas:
    Error:
      title: Error
      type: object
      properties:
        code:
          type: integer
          format: int32
        message:
          type: string

    Order:
      title: Order
      type: object
      required:
        - orderId
        - emissionDate
        - quantityPackages
        - serviceType
        - serviceDesc
        - productType
        - productDesc
        - productFamily
        - codeState
        - state
        - referenceOrder
        - isDangerousShipping
        - weight
        - volume
        - seller
        - buyer
        - package
      properties:
        orderId:
          type: string
          description: A unique identifier for order
        emissionDate:
```

```
        type: string
        description: Created date for order
      quantityPackages:
        type: integer
        format: int32
        description: Quantity packages in the order
      serviceType:
        type: string
        description: Service type code
      serviceDesc:
        type: string
        description: Service type description
      productType:
        type: string
        description: Product type code
      productDesc:
        type: string
        description: Product type description
      productFamily:
        type: string
        description: Product category code
      references:
        type: array
        description: Order reference list
        items:
          type: string
      observation:
        type: string
        description: The observation of order
      codeState:
        type: string
        description: State code
      state:
        type: string
        description: State description
      referenceOrder:
        type: string
        description: Reference of legacy order
      promiseDate:
        type: string
        description: Promise date
      isDangerousShipping:
        type: boolean
        description: Is dangerous shipping
      weight:
        type: string
        description: The weight of the order
      volume:
        type: string
```

```
          description: The volume of the order
        declaredValue:
          type: integer
          description: The declared value of the order
        seller:
          type: object
          description: The seller of the order
        buyer:
          type: object
          description: The buyer of the order
        cod:
          type: object
          description: The collect of delivery
        dd:
          type: object
          description: Devolution of documents
        insurance:
          type: object
          description: The insurance of order
        shipping:
          type: object
          description: The shipping cost of order
        package:
          type: array
          description: The package in the order
          items:
            type: string
        taxDocument:
          type: object
          description: The tax document of order
  securitySchemes:
    Blue-OAuth2:
      flows: {}
      type: oauth2
      description: Blue-OAuth2
    Blue-APIKey:
      type: apiKey
      description: Blue-APIKey
      name: apikey
      in: header
security:
  - Blue-OAuth2: []
```

Anexo 2: OpenAPI Specification para API Create-Emission

```
openapi: 3.0.1
info:
```

```yaml
  title: API Emission Orders
  description: API for the emission the Order
  contact:
    name: Ciclo de Vida de la OS
    email: andres.godoy@blue.cl
  version: 1.0.0
servers:
  - url: >-
      [REDACTED]{basePath}
    variables:
      enviroment:
        enum:
          - dev
          - qa
          - prod
        default: dev
      basePath:
        default: /ciclos/orders/v1
tags:
  - name: emission
    description: Emission Controller
paths:
  /{orderId}/validate:
    get:
      tags:
        - emission
      operationId: emissionController_getValidate
      summary: Validate if the OrderId is available for a new Emission
      parameters:
        - name: orderId
          in: path
          description: Enter the order number (9-10 digits)
          required: true
          schema:
            type: integer
            minimum: 9
            maximum: 10
      responses:
        '200':
          $ref: '#/components/responses/Validate'
        '400':
          $ref: '#/components/responses/BadRequest'
        '404':
          $ref: '#/components/responses/NotFound'
        '500':
          $ref: '#/components/responses/InternalServer'
  /{orderId}/activate:
    put:
      tags:
```

```yaml
      - emission
    operationId: emissionController_postActivate
    summary: Activate Emission the a Order
    parameters:
      - name: orderId
        in: path
        description: Enter the order number (9-10 digits)
        required: true
        schema:
          type: integer
          minimum: 9
          maximum: 10
    requestBody:
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Activate'
      required: true
    responses:
      '201':
        $ref: '#/components/responses/Response'
      '400':
        $ref: '#/components/responses/BadRequest'
      '404':
        $ref: '#/components/responses/NotFound'
      '500':
        $ref: '#/components/responses/InternalServer'
  /emission:
    post:
      tags:
        - emission
      operationId: emissionController_postEmission
      summary: Create New Emission the a Order
      requestBody:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Emission'
        required: true
      responses:
        '201':
          $ref: '#/components/responses/Response'
        '400':
          $ref: '#/components/responses/BadRequest'
        '500':
          $ref: '#/components/responses/InternalServer'
components:
  schemas:
    Emission:
```

```yaml
    required:
      - emissionDate
      - service
      - channel
      - isDangerousShipping
      - creationUser
      - seller
      - buyer
      - shipper
      - pickup
      - delivery
      - package
    type: object
    properties:
      emissionDate:
        type: string
      service:
        type: string
      channel:
        type: string
      isDangerousShipping:
        type: boolean
      creationUser:
        type: string
      observation:
        type: string
      references:
        type: array
        items:
          type: string
      declaredValue:
        type: number
      seller:
        type: object
        $ref: '#/components/schemas/Seller'
      buyer:
        type: object
        $ref: '#/components/schemas/Buyer'
      shipper:
        type: object
        $ref: '#/components/schemas/Shipper'
      shipping:
        type: object
        $ref: '#/components/schemas/Shipping'
      pickup:
        type: object
        $ref: '#/components/schemas/Address'
      delivery:
        type: object
```

```yaml
        $ref: '#/components/schemas/Address'
      complementary:
        type: array
        items:
          $ref: '#/components/schemas/ComplementaryServices'
      packages:
        type: array
        items:
          $ref: '#/components/schemas/Package'
    example:
      emissionDate: '2023-12-06T15:30:00Z'
      service: Standard
      channel: WEB
      isDangerousShipping: false
      creationUser: rod.grille
      observation: This is a sample emission
      references:
        - 1335485d4
        - 23va332e2
      declaredValue: 100.5
      seller:
        account: 78959482-12-8
        name: Seller Company
        identifier: 12345678-9
        phone: '912345678'
        email: seller@example.com
      buyer:
        name: John Dwayne
        identifier: 987654321-9
        phone: '+5698654321'
        email: buyer@example.com
      shipper:
        account: 80548613-1-80
        name: Shippers
      pickup:
        street: 123 Main St
        number: 456
        commune: Central Commune
        city: Cityville
        region: Regionville
        geolocalization:
          - 40.7128
          - -74.006
        complement: Near the park
        agency: 789
      delivery:
        street: 456 Side St
        number: 789
        commune: South Commune
```

```
        city: Cityville
        region: Regionville
        geolocalization:
          - 40.7128
          - -74.006
        complement: Across the street
        agency: 1011
      shipping:
        paymentType: PP
        tariffPlan: A32
        shippingCost: 1300
        freight: 1053
        discount: 200
        tax: 247
        payment:
          - method: TD
            date: '2023-12-10'
            currency: CLP
            amount: 1300
            transactionId: 123456789
      complementary:
        - serviceType: COD
          identifier: ABC123
          documentType: B
          amount: 2000
          collection:
            - method: TD
              date: '2023-12-15'
              amount: 2000
              transactionId: XYZ789
      packages:
        - size: M
          inspection: null
          height: 10
          length: 20
          width: 15
          weight: 25
          units:
            length: CM
            width: KG
  Activate:
    required:
      - emissionDate
      - service
      - channel
      - isDangerousShipping
      - creationUser
      - seller
      - buyer
```

```yaml
      - pickup
      - delivery
    type: object
    properties:
      emissionDate:
        type: string
      service:
        type: string
      channel:
        type: string
      isDangerousShipping:
        type: boolean
      creationUser:
        type: string
      observation:
        type: string
      references:
        type: array
        items:
          type: string
      declaredValue:
        type: number
      seller:
        type: object
        $ref: '#/components/schemas/Seller'
      buyer:
        type: object
        $ref: '#/components/schemas/Buyer'
      shipping:
        type: object
        $ref: '#/components/schemas/Shipping'
      pickup:
        type: object
        $ref: '#/components/schemas/Address'
      delivery:
        type: object
        $ref: '#/components/schemas/Address'
      complementary:
        type: array
        items:
          $ref: '#/components/schemas/ComplementaryServices'
      packages:
        type: array
        items:
          $ref: '#/components/schemas/Package'
    example:
      emissionDate: '2023-12-06T15:30:00Z'
      service: Standard
      channel: WEB
```

```yaml
        isDangerousShipping: false
        creationUser: rodrigo.grille
        observation: This is a sample emission
        references:
          - 1335485d4
          - 23va332e2
        declaredValue: 100.5
        seller:
          account: 78959482-12-8
          name: Seller Company
          identifier: 12345678-9
          phone: '912345678'
          email: seller@example.com
        buyer:
          name: John Dwayne
          identifier: 987654321-9
          phone: '+5698654321'
          email: buyer@example.com
        pickup:
          street: 123 Main St
          number: 456
          commune: Central Commune
          city: Cityville
          region: Regionville
          geolocalization:
            - 40.7128
            - -74.006
          complement: Near the park
          agency: 789
        delivery:
          street: 456 Side St
          number: 789
          commune: South Commune
          city: Cityville
          region: Regionville
          geolocalization:
            - 40.7128
            - -74.006
          complement: Across the street
          agency: 1011
        shipping:
          paymentType: PP
          tariffPlan: A32
          shippingCost: 1300
          freight: 1053
          discount: 200
          tax: 247
          payment:
            - method: TD
```

```
            date: '2023-12-10'
            currency: CLP
            amount: 1300
            transactionId: 123456789
      complementary:
        - serviceType: COD
          identifier: ABC123
          documentType: B
          amount: 2000
          collection:
            - method: TD
              date: '2023-12-15'
              amount: 2000
              transactionId: XYZ789
      packages:
        - size: M
          inspection: null
          height: 10
          length: 20
          width: 15
          weight: 25
          units:
            length: CM
            width: KG
  Seller:
    type: object
    required:
      - account
      - name
      - phone
      - email
    properties:
      account:
        type: string
      name:
        type: string
      identifier:
        type: string
      phone:
        type: string
      email:
        type: string
  Buyer:
    type: object
    required:
      - name
      - phone
      - email
    properties:
```

```
      name:
        type: string
      identifier:
        type: string
      phone:
        type: string
      email:
        type: string
  Shipper:
    type: object
    required:
      - account
    properties:
      account:
        type: string
      name:
        type: string
  Shipping:
    type: object
    properties:
      paymentType:
        type: string
      tariffPlan:
        type: string
      shippingCost:
        type: number
      freight:
        type: number
      discount:
        type: number
      tax:
        type: number
      payment:
        type: array
        items:
          $ref: '#/components/schemas/Payment'
  Address:
    type: object
    required:
      - street
      - number
      - commune
      - region
    properties:
      street:
        type: string
      number:
        type: integer
      commune:
```

```
        type: string
      city:
        type: string
      region:
        type: string
      location:
        type: string
      geolocalization:
        type: object
        $ref: '#/components/schemas/Geo'
      complement:
        type: string
      agency:
        type: integer
  Payment:
    type: object
    properties:
      method:
        type: string
      date:
        type: string
      currency:
        type: string
      amount:
        type: number
      transactionId:
        type: number
  ComplementaryServices:
    type: object
    properties:
      serviceType:
        type: string
      identifier:
        type: string
      documentType:
        type: string
      amount:
        type: number
      collection:
        type: array
        items:
          $ref: '#/components/schemas/Collection'
  Collection:
    type: object
    properties:
      method:
        type: string
      date:
        type: string
```

```
      amount:
        type: number
      transactionId:
        type: string
      currency:
        type: string
  Package:
    type: object
    required:
      - inspection
    properties:
      size:
        type: string
      inspection:
        type: object
        $ref: '#/components/schemas/Inspection'
  Inspection:
    type: object
    properties:
      height:
        type: number
      length:
        type: number
      width:
        type: number
      weight:
        type: number
      units:
        type: object
        $ref: '#/components/schemas/Units'
  Units:
    type: object
    properties:
      length:
        type: string
      width:
        type: string
  Geo:
    type: array
    maxItems: 2
    minItems: 2
    items:
      type: number
      format: double
      minimum: -180
      maximum: 180
  Error:
    type: object
    properties:
```

```
      message:
        type: string
responses:
  Response:
    description: Emission Succedfull
    content:
      application/json:
        schema:
          type: object
          properties:
            orderId:
              type: string
            message:
              type: string
        example:
          orderId: 6610670614
          message: Orden de servicio emitida correctamente
  Validate:
    description: Validate Order
    content:
      application/json:
        schema:
          type: object
          properties:
            orderId:
              type: string
            size:
              type: string
              enum:
                - S
                - M
                - L
                - XL
            message:
              type: string
        example:
          orderId: '6610670614'
          size: S
          message: Orden de servicio disponible
  BadRequest:
    description: Bad Request
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Error'
        example:
          message: La Orden no tiene el formato correcto
  NotFound:
    description: Not Found
```

```
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
          examples:
            sample 1:
              value:
                message: Orden de servicio no disponible para la emisión
            sample 2:
              value:
                message: No es posible acceder al recurso
    InternalServer:
      description: Internal Server Error
      content:
        application/json:
          schema:
            $ref: '#/components/schemas/Error'
          example:
            message: No se pudo ejecutar la operación
  securitySchemes:
    Blue-OAuth2:
      flows: {}
      type: oauth2
      description: Blue-OAuth2
    Blue-APIKey:
      type: apiKey
      description: Blue-APIKey
      name: apikey
      in: header
security:
  - Blue-OAuth2: []
  - Blue-APIKey: []
```